<MINESOTA STATE TAX REGISTRATION APP REFACTORING>

OVERALL REPORT

VERSION <1.0>

<Alexandros Milonakis AM:3045 Konstantinos Zoulias AM:3155>

TABLE OF CONTENTS

INTRODUCTION

Write a brief introduction about the objectives of this phase of the project.

REFACTORED DESIGN

USE CASES

Specify the use cases of the application.

Use case ID	UC1
Actors	Minnesota State
Pre conditions	1. There must be a file containing the taxpayer's data.
Main flow of events	1. The use case starts when the user loads a taxpayer.
Alternative flow 1	1. If the user doesn't give 9 digits 1.1 Exception must have 9 digits
Alternative flow 2	1.If the file the user gives doesn't exist 1.1 Exception doesn't exist
Alternative flow 3	1.if the tax registration number exists in the system 1.1 Exception already exists
Post conditions	The tax registration number is displayed in the GUI.

Use case ID	UC2
Actors	Minnesota State
Pre conditions	The taxpayer's data must have been loaded.
Main flow of events	1. The use case starts when the user selects a taxpayer
Alternative flow 1	1. If the tax registration number hasn't been loaded 1.1 Exception isn't loaded
Alternative flow 2	1. If the tax registration number is wrong 1.1 Exception must give a registration number
Post conditions	The application opens a new window containing the taxpayers information

Use case ID	UC3
Actors	Minnesota State
Pre conditions	1. Taxpayers data must have been previously loaded
Main flow of events	1. The use case starts when the user wants to delete a taxpayer.
Alternative flow 1	1.Does nothing if incorrect registration number
Post conditions	The tax registration number deleted from the GUI.

Use case ID	UC4
Actors	Minnesota State
Pre conditions	1. There must be a file containing the taxpayer's data and the user must have loaded said information.
Main flow of events	1. The use case starts after the user wants to add a new receipt for the current taxpayer. 2.The user must type the receipt ID, date, the kind of the receipt, the amount of money. 3. .The user must type the company's(that the receipt came from) information (name, city it is based in, the street and the number)
Alternative flow 1	1. If said information doesn't conform with the system rules the receipt isn't added to the system
Post conditions	The receipt is added to the taxpayer's information.

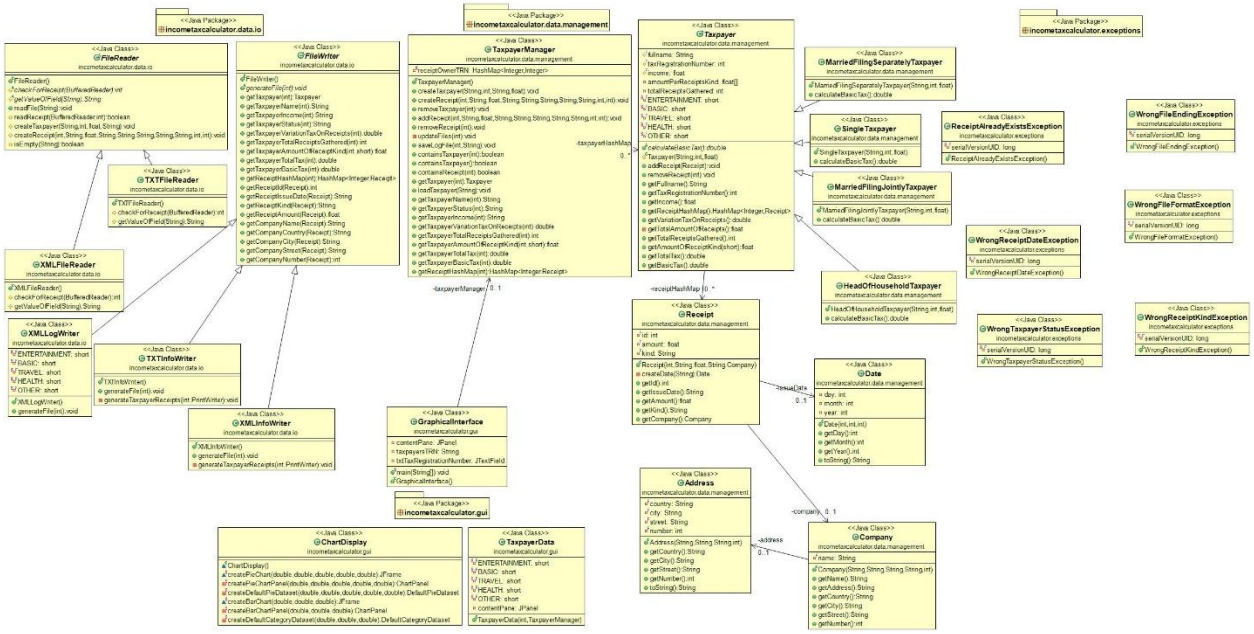
Use case ID	UC5
Actors	Minnesota State
Pre conditions	1. There must be a file containing the taxpayer's data and the user must have loaded said information.
Main flow of events	1. The use case starts after the user wants to delete a receipt for the current taxpayer. 2.The user must type the receipt ID that he wants deleted.
Alternative flow 1	1. Does nothing if the receipt Id doesn't exist.
Post conditions	The receipt is deleted from the taxpayer's information.

Use case ID	UC6
Actors	Minnesota State
Pre conditions	1. There must be a file containing the taxpayer's data and the user must have loaded said information.
Main flow of events	1. The use case starts when the user wants to view a report about the taxpayer's information. 2. The application pops up a window containing a bar chart with information about the tax that the taxpayer must pay. 3. The application pops up a window containing a pie chart containing information about the receipts of the user.
Post conditions	-

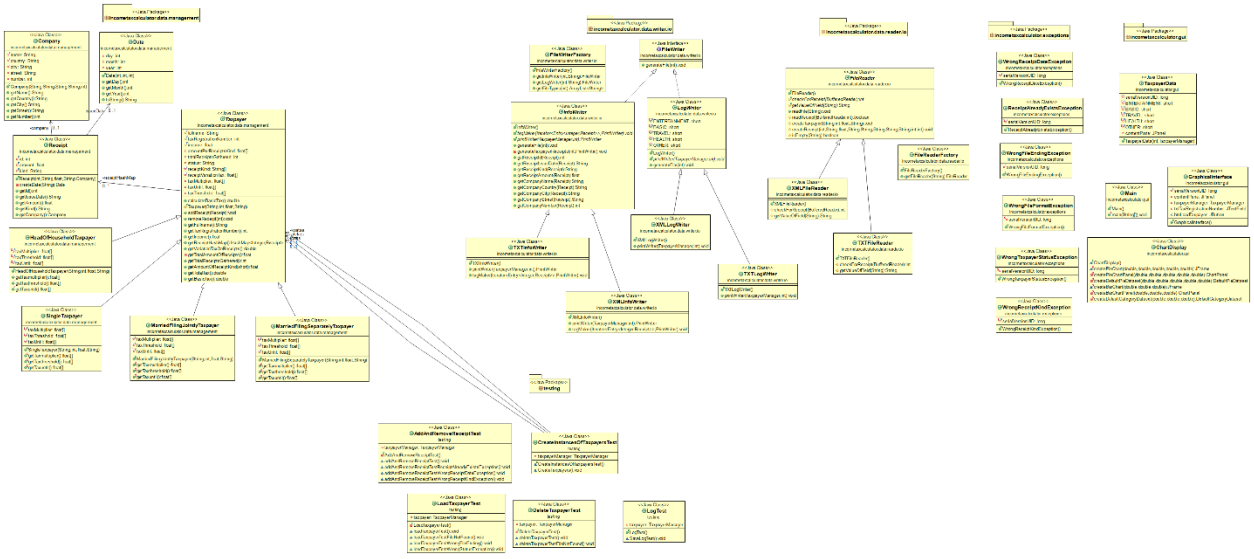
Use case ID	UC6
Actors	Minnesota State
Pre conditions	1. There must be a file containing the taxpayer's data and the user must have loaded said information.
Main flow of events	1. The use case starts after the user wants to save the information about the current taxpayer.
Post conditions	The information is saved in the existing txt and xml files.

ARCHITECTURE

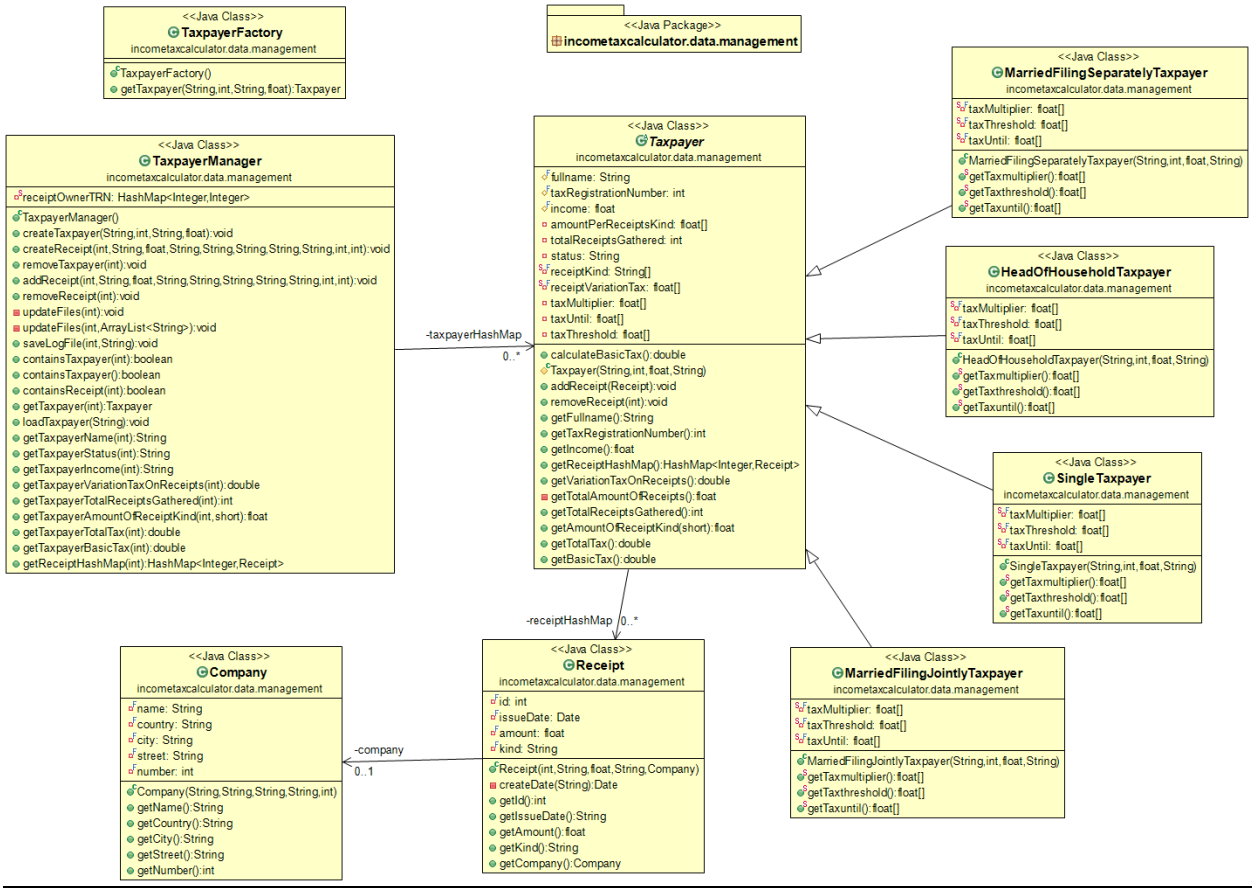
Original Application UML:



Final Application UML:



DETAILED DESIGN



❖ For the Data Management package

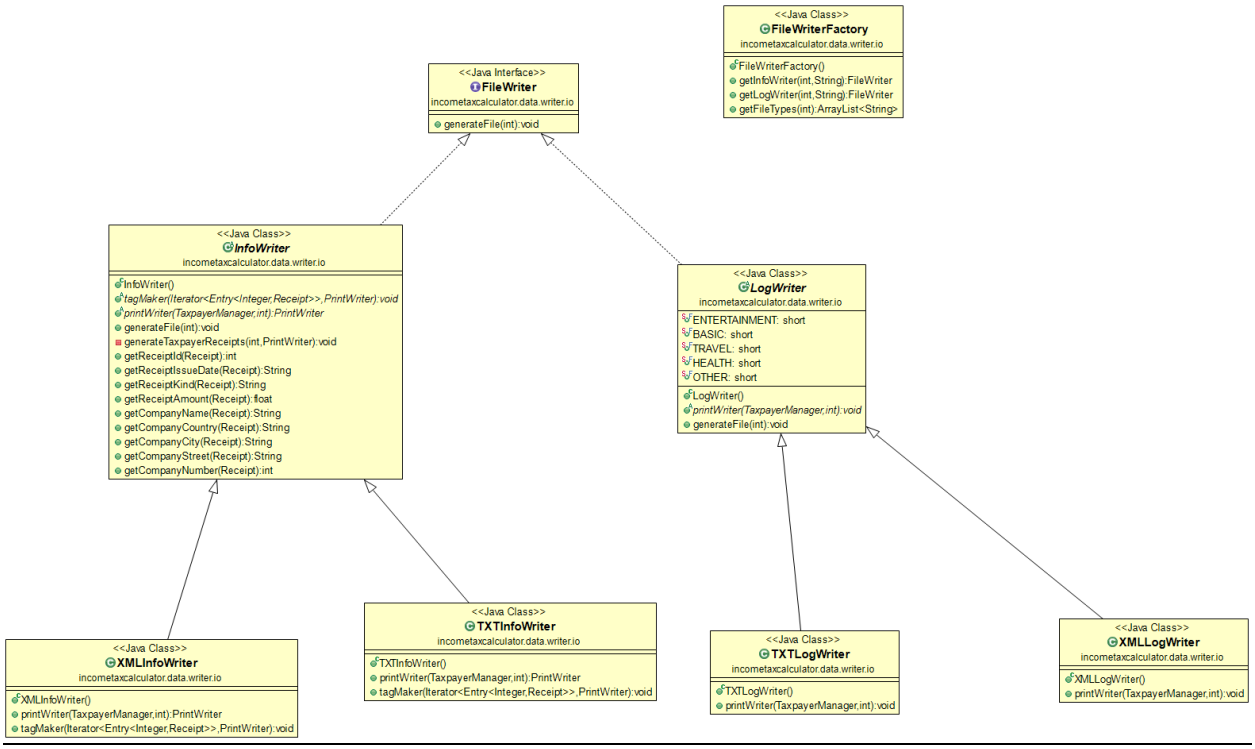
- We removed the address method to get rid of unnecessary complexity. Instead of the address we relocated its fields to the company class and without creating new methods we changed the already existing getters to match our new fields.

- Then we removed the complex conditional logic from the addReceipt(), removeReceipt() and getVariationTaxOnReceipts() methods by implementing an simple for loop.

- Continuing we addressed the problem of duplicate code in our subclasses (different taxpayer classes) by refactoring the abstract method calculateBasicTax() to implement the code for the calculation of the basic tax while getting the required information from the Taxpayer subclasses about the required fields.(Arrays containing information about the tax multipliers)

- Taxpayer Manager Class:

  We refactored the Taxpayer manager class. For starters we moved the conditional logic that creates different kind of taxpayer objects to our new factory class(TaxpayerFactory)



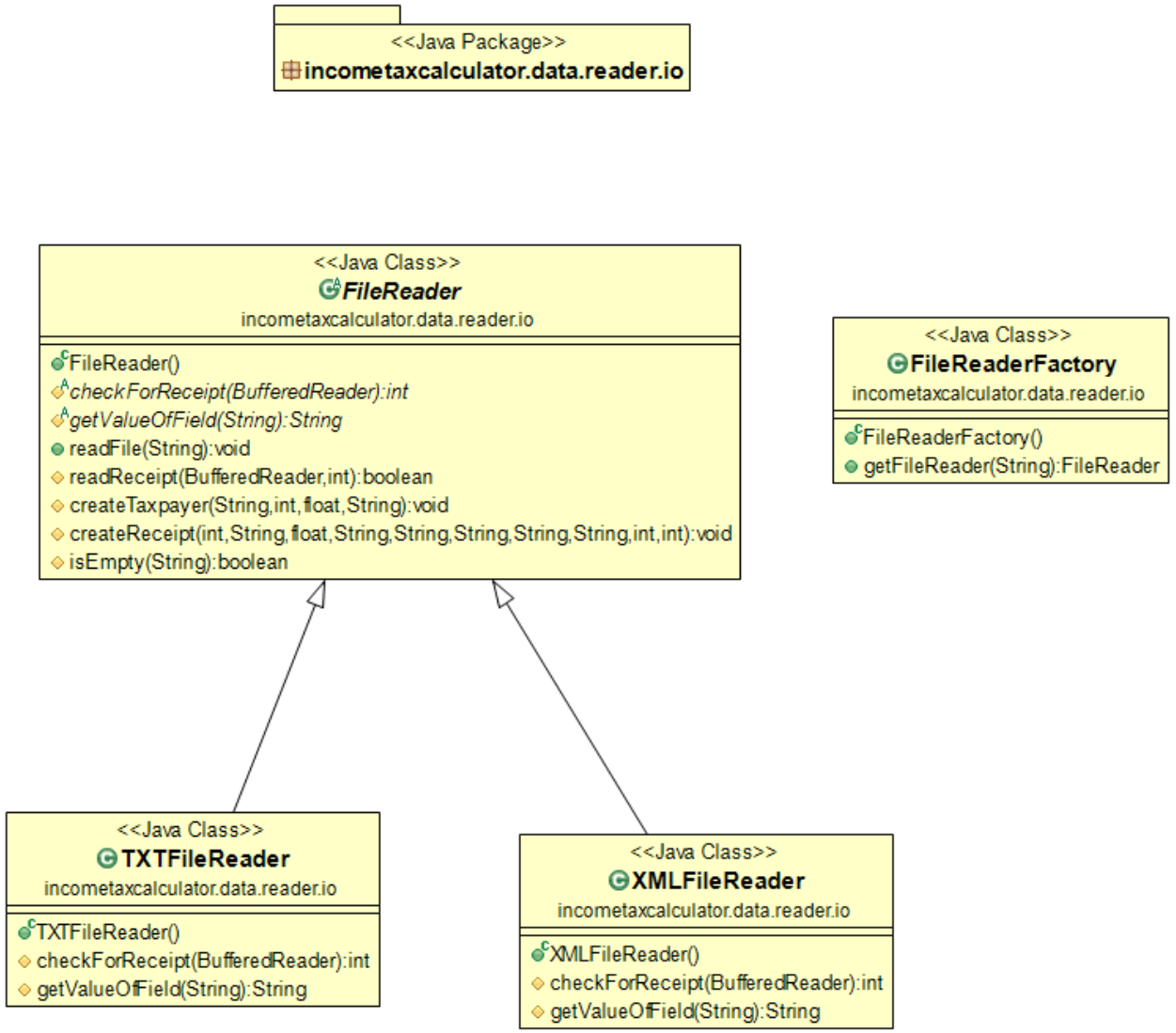
- Then, we to easier the load of this class even more we moved the conditional logic that updates and saves a log to our FileWriterFactory class (responsible for creating Writer type objects for creating/saving/updating the taxpayer's files

1. For the Writer Package:

   - We created an InfoWriter class and a LogWriter class to address the problem in duplicate code in both XML Info & Log Writer and TXT Info & Log Writer correspondingly and moved the duplicate code to the parent class (Info Writer and Log Writer)

   - Because we noticed the similar methods for its of the writer classes we created an Interface for the writer class to encapsule those methods

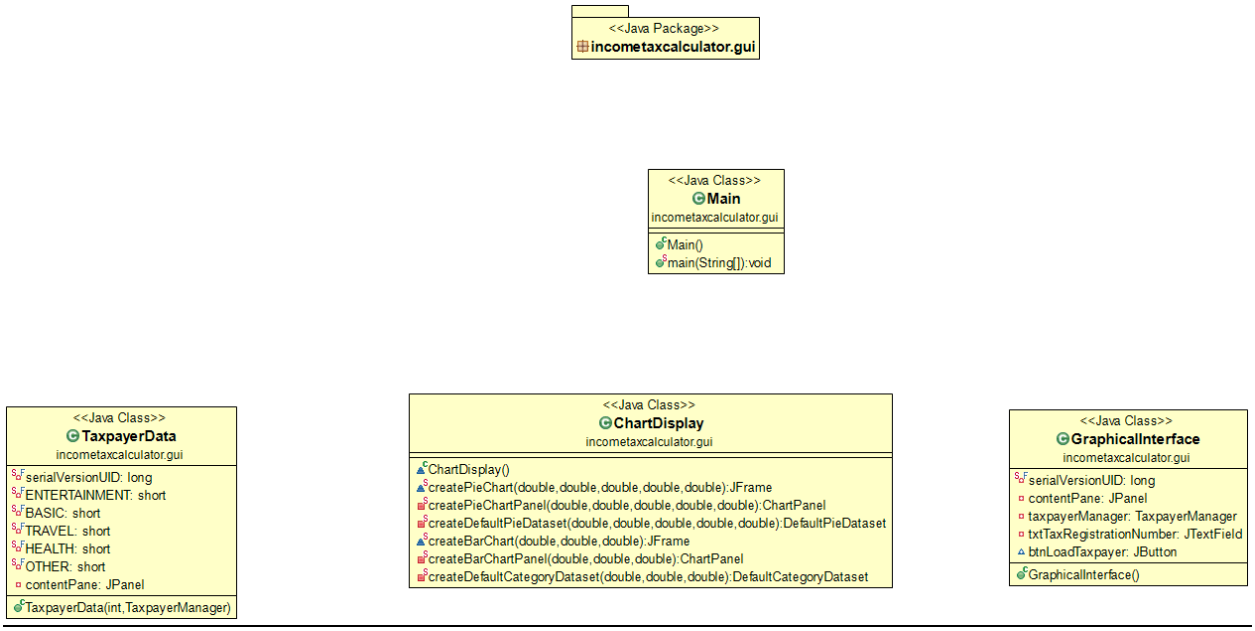
- Lastly we created a Writer Factory to create writer type objects for us to handle



2. For the Reader Package:

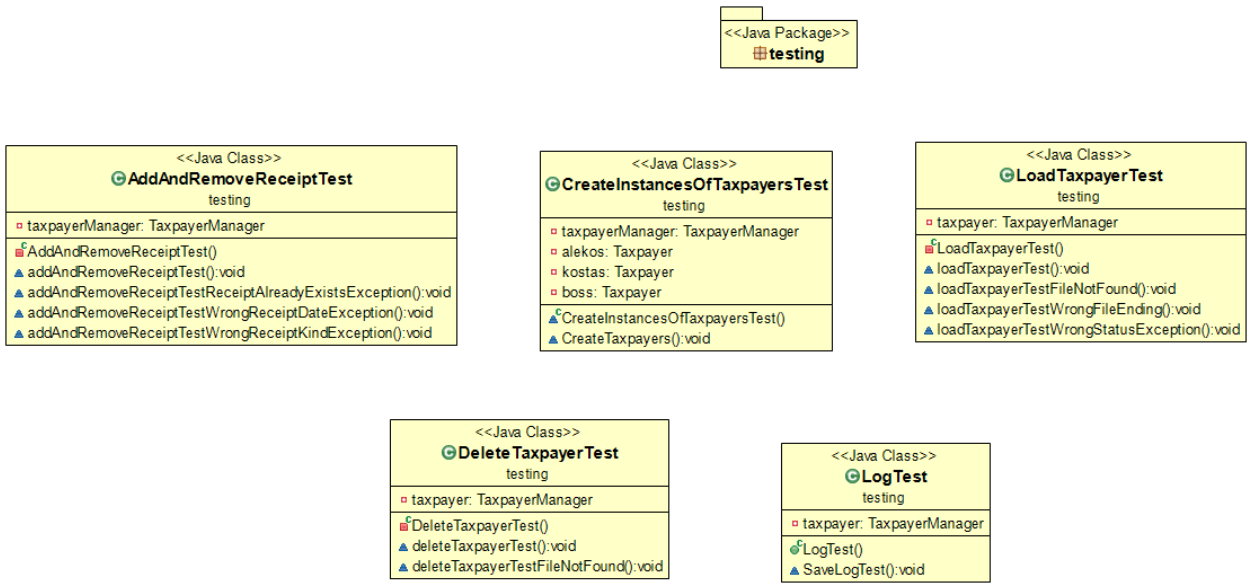
   - We acted with a similar mindset on Reader package we took the similar code from each reader type class and we incorporated them in an abstract class named Reader

   - We also created a Reader Factory to create reader type objects for us to handle .



3. GUI Package:

For the GUI package to make the application easier to use we refactored the Graphical Interface class. We changed the way the application loads a file(We made the user able to select a file from his own computer. Secondly we changed the way the user selects a TRN(We made the list clickable and when the user clicks the select button it loads the highlighted taxpayer's data



4. Testing Package

For the purpose of testing our code we created 5 Junit tests the run through the code and make sure everything is working correctly. Unfortunately we weren't able to test the GUI with any test.

CLASSES RESPONSIBILITIES AND COLLABORATIONS (CRC CARDS)

- Package : incometax.calculator.data.management

Class Name: Company	
Responsibilities	**Collaborations**
➢ The Company class creates a company type object with all the required information about it.	▪ TaxpayerManager class

Class Name: Date	
Responsibilities	**Collaborations**
➢ The Date class takes the information(day, month , year) and returns it in the form of a date (day/month/year)	▪ Company class

Class Name: HeadOfHouseholdTaxpayer	
Responsibilities	**Collaborations**
➢ This class is instantiates a taxpayer who is the Head of the household.	▪ Taxpayer Class ▪ TaxpayerFactory Class

Class Name: MarriedFillingJointlyTaxpayer	
Responsibilities	**Collaborations**
➢ This class is responsible for instantiating a taxpayer who is the married and filling jointly.	▪ Taxpayer Class ▪ TaxpayerFactory Class

Class Name: MarriedFillingSeparatelyTaxpayer	
Responsibilities	**Collaborations**
➢ This class is responsible for instantiating a taxpayer who is the married and filling separately.	▪ Taxpayer Class ▪ TaxpayerFactory Class

Class Name: Receipt	
Responsibilities	**Collaborations**
➢ This class is responsible for creating a receipt type object.	▪ Date Class ▪ Company Class ▪ TaxpayerManager Class

Class Name: SingleTaxpayer	
Responsibilities	**Collaborations**
➢ This class is responsible for instantiating a taxpayer who is the single.	▪ Taxpayer Class ▪ TaxpayerFactory Class

Class Name: Taxpayer	
Responsibilities	**Collaborations**
➢ This abstract class is responsible for creating a taxpayer type object. ➢ This class calculates the tax of each taxpayer ➢ This class manages the receipts of each taxpayer	▪ Receipt class ▪ (HeadOfHouseholdTaxpayer , MarriedFillingSeparatelyTaxpayer , MarriedFillingJointlyTaxpayer , SingleTaxpayer) taxpayer classes.

Class Name: TaxpayerFactory	
Responsibilities	**Collaborations**
➢ This class is responsible for instantiating a taxpayer based on which category of taxpayers he belongs to.	▪ Taxpayer class ▪ (HeadOfHouseholdTaxpayer , MarriedFillingSeparatelyTaxpayer , MarriedFillingJointlyTaxpayer , SingleTaxpayer) taxpayer classes.

Class Name: TaxpayerManager	
Responsibilities	**Collaborations**
➢ This class is responsible for managing all taxpayer type objects. ➢ This class is responsible for managing all receipt type objects. ➢ It is also responsible for interacting with the files (update and save file functions)	▪ Taxpayer class ▪ TaxpayerFactory class ▪ Receipt class ▪ FileReader class ▪ FileWriter class ▪ TaxpayerData class ▪ GraphicalInterface class

- Package : incometaxcalculator.data.reader.io

Class Name: FileReader	
Responsibilities	**Collaborations**
➢ This class is responsible for parsing through the requested file ➢ This class interacts with the taxpayer manager class in order to update the database with the input from the requested file	▪ TaxpayerManager class

Class Name: FileReaderFactory	
Responsibilities	**Collaborations**
➢ This class creates a FileReader type object depending on ιτς file type	▪ FileReader Interface ▪ XMLFileReader class ▪ TXTFileReader class

Class Name: TXTFileReader	
Responsibilities	**Collaborations**
➢ This class is responsible for parsing through a TXT File.	▪ FileReader interface ▪ FileReaderFactory class

Class Name: XMLFileReader	
Responsibilities	**Collaborations**
➢ This class is responsible for parsing through an XML File.	▪ FileReader Interface ▪ FileReaderFactory class

- Package : incometaxcalcualtor.data.writer.io

Class Name: FileWriterFactory	
Responsibilities	**Collaborations**
➢ This class is responsible for creating the correct Writer type object according to what the user requested.	▪ FileWriter Interface ▪ InfoWriter class ▪ LogWriter class ▪ TXTInfoWriter class ▪ XMLInfoWriter class ▪ TXTLogWriter class ▪ XMLLogWriter class

Class Name: InfoWriter	
Responsibilities	**Collaborations**
➢ This class is responsible to write information (update, add) about the receipts of the taxpayer.	▪ FileWriter class ▪ FileWriterFactory class ▪ TXTInfoWriter ▪ XMLInfoWriter

Class Name: LogWriter	
Responsibilities	**Collaborations**
➢ This class is responsible to write the taxpayers information.	▪ FileWriter class ▪ FileWriterFactory class ▪ TXTLogWriter ▪ XMLLogWriter

Class Name: TXTInfoWriter	
Responsibilities	**Collaborations**
➢ This class implements the InfoWriter class on a TXT file.	▪ FileWriter class ▪ FileWriterFactory class ▪ InfoWriter class

Class Name: TXTLogWriter	
Responsibilities	**Collaborations**
➢ This class implements the LogWriter class on a TXT file.	▪ FileWriter class ▪ FileWriterFactory class ▪ LogWriter class

Class Name: XMLInfoWriter	
Responsibilities	**Collaborations**
➢ This class implements the InfoWriter class on a XML file.	▪ FileWriter class ▪ FileWriterFactory class ▪ InfoWriter class

Class Name: XMLLogWriter	
Responsibilities	**Collaborations**
➢ This class implements the LogWriter class on a XML file.	▪ FileWriter class ▪ FileWriterFactory class ▪ LogWriter class

▪

- Package : incometaxcalcualtor.gui

Class Name: ChartDisplay	
Responsibilities	**Collaborations**
➢ This class is responsible for creating the charts containing the taxpayers data.	

Class Name: GraphicalInterface	
Responsibilities	**Collaborations**
➢ This class is responsible for creating the graphical interface for our application.	▪ TaxpayerManager ▪ Main

Class Name: Main	
Responsibilities	**Collaborations**
➢ This class is responsible for initiating the program.	▪ GraphicalInterface

Class Name: TaxpayerData	
Responsibilities	**Collaborations**
➢ This class is responsible for handling the taxpayer's data within the GUI .	▪ TaxpayerManager class ▪ Receipt class

- Package: incometaxcalcualtor.exceptions
  - This package has classes to handle the exceptions throughout our code in any given circumstances. These classes (ReceiptAlreadyExistsException.java, WrongFileEndingException.java, WrongFileFormatException.java, WrongReceiptDateException.java, WrongReceiptKindException.java, WrongTaxpayerStatus.java) are handling exception not already handled by the java libraries and would be detrimental to our code if they didn't exist. They are interacting with every other package in various methods when necessary.